(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 037 147 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
20.09.2000 Bulletin 2000/38

(51) Int Cl.[7]: **G06F 9/46**

(21) Application number: **99303936.1**

(22) Date of filing: **20.05.1999**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **15.03.1999 EP 99301952**

(71) Applicant: **BRITISH TELECOMMUNICATIONS public limited company**
London EC1A 7AJ (GB)

(72) Inventor: **The designation of the inventor has not yet been filed**

(74) Representative: **Geary, Stuart Lloyd et al**
Venner, Shipley & Co.,
20 Little Britain
London EC1A 7DH (GB)

(54) Resource scheduling

(57) A method of administering resource utilisation in a computer comprises running a first process (13, 14, 15) to make a reservation for access to a resource in dependence on a resource requirement communication from an application (21) and running a second process (16, 17, 19) to grant requests for access to said resource from said application in dependence on said reservation. A further process (12) provides a common interface between the first process (13, 14, 15) for each resource and the application (21). The further process (12) converts high-level abstract resource requirement definitions into formats applicable to the first process (13, 14, 15) for the resource in question. The processes are preferably implemented at methods of software objects.

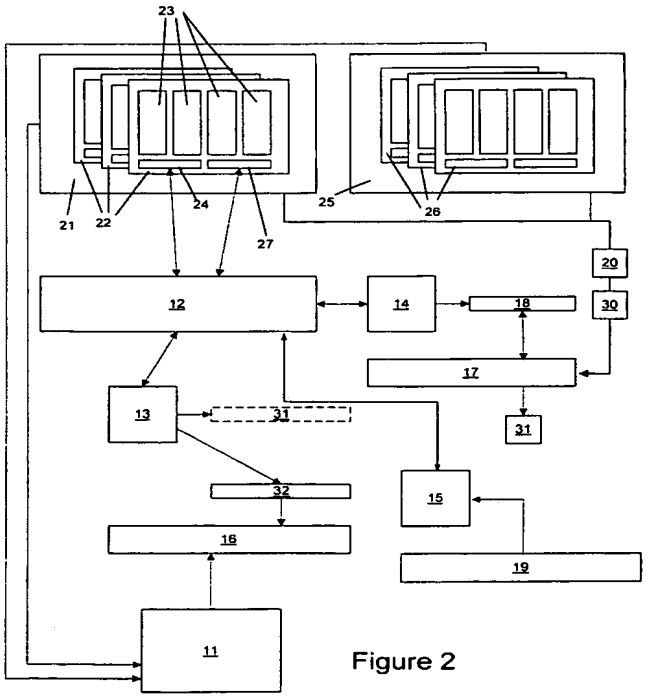

Figure 2

EP 1 037 147 A1

Description

[0001] The present invention relates to resource scheduling in a computer.

[0002] Enterprise level general-purpose operating systems, both workstation and server, are traditionally designed for efficient resource utilisation. Applications share a common set of resources, which are initially scheduled according to their priority and then in a fair weighted manner. This approach results is both an efficient and fair division of resources to user-level applications. However, certain applications, in particularly multimedia processing applications, are inherently sensitive to resource availability. As a result, the use of priority-based resource federation policies often results in such applications failing to meet their processing goals.

[0003] Conventionally, relatively complex processing is required of the scheduling process at the stage where a process is granted access to a resource. The present invention enables a reduction in the complexity of the processing at the access grant stage by providing for reservation of resources in advance.

[0004] According to the present invention, there is provided a method of administering resource utilisation in a computer, the method comprising running a first process to make a reservation for access to a resource in dependence on a resource requirement communication from an application process and running a second process to grant requests for access to said resource from said application process in dependence on said reservation. The reservation request may be made by an application *per se* or a sub-process or component thereof. Furthermore, access may be granted to any process of an application or for a particular sub-unit of the application for which the reservation request was made.

[0005] Making reservations and granting access at application level assists in embodiments of the present invention intended to operate with legacy code. However, where all applications are written for an operating system employing a scheduling method according to the present invention, component or thread level reservation and access grant is preferred.

[0006] Preferably, a method according to the present invention comprises:

instantiating a scheduling object having a method or methods for processing reservation requests for a plurality of resources and initiating resource specific reservation processing;
instantiating a reservation object having a method or methods for making reservations for access to a resource;
invoking a method of the scheduling object from an application process, said method taking a first resource access requirement definition as a parameter;
invoking, from said method of the scheduling object with a second resource access requirement definition as a parameter, a reservation method (said first process) of the reservation object to make a reservation for the application process;
running a resource specific scheduling process (said second process) to grant access to a resource in dependence on the reservation made by the reservation object; and
utilising said resource for the purposes of said application process.

[0007] Preferably, the method of the scheduling object translates the first resource requirement definition into the second resource requirement definition. The advantage of this is that the resource request can be defined by a programmer in a hardware independent manner and the scheduling object for a particular platform can translate that programmer-produced resource request in a potentially non-linear manner on the basis of the properties of the particular platform.

[0008] A method according to the present invention can be applied to the allocation of cpu time, access to mass storage devices, e.g. hard disk drives, optical disk drives and the like, and memory management.

[0009] Preferably in the case of mass storage access, said first process comprises allocating one or more "lottery ticket" numbers to said application or a process thereof in dependence on the second resource access requirement definition and storing requests for access to a mass storage device from application processes;

generating substantially randomly a lottery ticket number; and
if no application process has been allocated said lottery ticket number then passing on to a mass storage device driver process the stored request for access from an application process selected on the basis of a predetermined prioritisation criterion else passing on to a mass storage device driver process a stored request for access from an application process to which said lottery ticket number was allocated.

[0010] The majority of multi-tasking operating systems, including Windows NT, Unix and Linux, use a scheduling algorithm known as Round-Robin. At the end of a time slot, or when a thread has revoked its time slice, the dispatcher takes off the first thread which is waiting on the highest priority queue (some queues will be empty). Additional techniques are used to prevent starvation by dynamically increasing the priority of long-waiting threads.

[0011] According to the present invention, there is also provided a method of scheduling access to a cpu which may be implemented in a method of administering resource utilisation in a computer according to the present invention, the method comprising the steps of:

generating a one-dimensional reservation request pattern;
merging the reservation request pattern with a one-dimensional cpu access control pattern, representing empty cpu access time slots and reserved cpu access time slots, without substantially disturbing either the reservation request pattern or the reserved cpu access time slots in the reservation request pattern.

[0012] The reservation request pattern may be shorter than the cpu access control pattern, in which case the reservation request pattern is effectively extended by repetition in the merging process.

[0013] Preferably, said merging step comprises relocating a non-empty time slot element of the reservation request pattern or the cpu access control pattern such that the patterns can be merged without any reserved cpu access time slot elements being deleted or overwritten. More preferably, the relocated non-empty time slot element is relocated by an amount defined in said time slot element. Both forwards and backwards shifts or shift limits may be included in each element.

[0014] According to the present invention, there is provided a method of granting access to a cpu which may be implemented in a method of administering resource utilisation in a computer according to the present invention, the method comprising:

generating a one-dimensional cpu access control pattern, each element of which relates to a quantum of cpu access time; and
at the end of a quantum of cpu access time:

granting access to any pending processes having a priority greater than a predetermined level; and then
if the next pattern element is empty then granting access to a pending process meeting a predetermined prioritisation (e.g. Round-Robin) criterion else granting access to a process identified in the pattern element.

[0015] Preferably, an entry for the process in any of a plurality of different priority queues will be searched for and access will be granted in respect of the entry for the process having the highest priority. Alternatively, access is granted to the process identified in the pattern element when there is not a populated process queue having a higher priority than the queue in which said process is present.

[0016] Preferably, the a one-dimensional cpu access control pattern is generated by a method of scheduling access to a cpu according to the present invention.

[0017] Embodiments of the present invention will now be described, by way of example, with reference to the accompanying drawings, in which:-

Figure 1 is a block diagram of a general purpose computer;
Figure 2 is a block diagram of software components embodied in a computer employing resource scheduling according to the present invention;
Figure 3 illustrates part of a cpu time reservation method according to the present invention;
Figure 4 is a flow chart illustrating the operation of the cpu secondary scheduler of Figure 2; and
Figure 5 is a flow chart illustrating an alternative manner of operation of the cpu scheduler of Figure 2.

[0018] Referring to Figure 1, a general purpose computer comprises a CPU 1, RAM 2, ROM 3, a video interface card 4, a hard disk drive controller 5, a keyboard interface 6 and a mouse interface 7 all interconnected by a combined data and address bus 8. A video monitor 9 is connected to the output of the video interface card 4. A hard disk drive 10 is connected to the hard disk drive controller 5. A keyboard 11 is connected to the input of the keyboard interface 6 and a mouse 12 is connected to the input of the mouse interface 7.

[0019] The general purpose computer is operating under the control of a multi-tasking operating system. The operating system may be a known operating system such as Windows NT, Unix or Linux but with the resource scheduling modified to operate as described below.

[0020] Referring to Figure 2, in order to provide resource sharing the operating system comprises a dispatcher component 11, a primary scheduler component 12, a cpu reservation component 13, a hard disk drive reservation component 14, a memory reservation component 15, a cpu secondary scheduler 16, a hard disk drive secondary scheduler 17 and a memory balance manager 19.

[0021] The dispatcher component 11 maintains queues of threads awaiting servicing by the CPU 1. A thread may have a priority value between 0 and 31, 31 being the highest priority, and the dispatcher component 11 maintains a separate queue for each priority. Thus, all the threads with priority 0 are in one queue, all the threads with priority 1 are in another queue and so on. Within each queue, new threads are added to the tail of the queue which operates on a FIFO principle.

[0022] The cpu secondary scheduler 16 is responsible for granting access to the CPU 1 to threads in a manner that will be described in more detail below.

[0023] The hard disk drive secondary scheduler 17 runs a lottery to determine access to the hard disk drive 10.

[0024] The primary scheduler 12 translates high-level abstract resource reservation requests, made by or on behalf of application components 22, into a form suitable for use by the reservation components 13, 14, 15. The reservation components 13, 14, 15 attempt to reserve the requested resources.

[0025] A guaranteed service application 21 for use with the present embodiment, consists of a plurality of a components 22 that are instantiated as necessary. Each component 22 comprises at least one thread 23. The constructor 24 of each component 22 includes a procedure which calls cpu, hard disk drive and memory reservation methods of the primary scheduler 12. The component 22 indicates its level of need for a resource by passing the application's process ID (which is unique to the application 21), an indication of the amount of resource required as parameters when calling the reservation methods and optionally a duration for the reservation. Alternatively, the application 21 itself or a management entity, can make a reservation on behalf of the guaranteed service application 21.

[0026] A best effort service application 25 consists of a plurality of a components 26 that are instantiated as necessary. The best effort service application's components 26 do not make resource reservations or have reservations made for them.

[0027] Considering now the case of a request for cpu time, when the cpu reservation method of the primary scheduler 12 is invoked by an application component's constructor 24, the cpu reservation method maps a percentage of resource time, received as a reservation parameter, into a component "signature". The component signature comprises a one-dimensional array of up to 256 elements, each element of which is a record comprising fields for the application's process ID for the component 22, a reservation ID, the forwards flexibility and the backwards flexibility. Initially, the fields of the elements of the array are empty. The fields of selected elements of the array are then filled using the parameters of the call made by the component 22. For instance, if the percentage value in the call is 10%, every tenth element of the array is filled and if the percentage value us 20%, every fifth element of the array is filled in. This mapping generally aims to achieve the finest grained reservation signature possible.

[0028] Once the component signature has been completed, the cpu reservation component 13 is instantiated on demand by the primary scheduler 12. When the cpu reservation component 13 is instantiated, the primary scheduler 12 forwards the component signature, application and reservation Ids and any duration to the cpu reservation component 13 via a method invocation.

[0029] The cpu reservation component 13 then tries to merge the component signature into a copy 31 of a master signature 32. The component signature is replicated to fill the full lengths of the signature arrays 31, 32 used by the cpu reservation component 13 and the cpu secondary scheduler 16. For example a reservation signature of 25% maps onto a 1 in 4 slot signature which is repeated 64 times. If any elements of the component signature clashes with an existing reservations (see Figure 3), the cpu reservation component 13 tries moving the clashing elements of the component signature according to their forwards and backwards flexibility values and moving the clashing elements of the copy 31 of the master signature 32, according to their backwards and forwards flexibilities, until the component signature fits into the copy 31 of the master signature 32. If this cannot be achieved an exception is raised and communicated back to the component 22 via the primary scheduler 12. If merging is achieved, the master signature 32 is updated according to the modified copy 31 and this information is communicated back to the primary scheduler 12. The primary scheduler 12 then returns to reservation ID to the calling component 22.

[0030] The cpu reservation component 13 always maintains at least a predetermined minimum number of elements of the master signature 32 free for the set of best effort applications 25. Each element of the master signature 32 corresponds to one cpu access time slice, e.g. 20ms. The cpu reservation component 13 automatically deletes reservations, requested with a duration parameter, that have time-expired.

[0031] The cpu secondary scheduler 16 uses the master signature 32 cyclically to allocate cpu time to components 22. The use of allocated cpu time by the threads 23 within a component 22 is the responsibility of the application programmer.

[0032] Priority based scheduling is used to schedule threads 23 which are very high priority threads (above priority level 15) that are essential to the integrity of the system. This means that threads 23 associated with time-critical operating system tasks, do not have to undergo the process of reservation and admission, as described above. The master signature 32 is used specifically for priority levels 0-15. The reason for this approach is to make possible support for legacy operating system code and services that are essential to the stability of the system.

[0033] The cpu secondary scheduler 16, uses the standard Round-Robin scheduling algorithm for threads with a priority greater than 15. Generally, these threads will not require a complete time-slice and are therefore negligible in terms of cpu access time.

[0034] Referring to Figure 4, the cpu secondary scheduler 16 repeatedly performs the following process. The cpu secondary scheduler 16 determines whether a new thread is pending in the dispatcher 11 (step s1). If a new thread is pending, the cpu secondary scheduler 16 determines whether a thread is currently running (step s2). If a thread is currently running, it determines whether the new pending thread has a higher priority than the running thread (in this context a guaranteed service thread has "priority" over a best-effort service thread with the same priority) (step s3). If the new thread's priority is higher, the cpu secondary scheduler 16 pre-empts the running thread, dispatches the new thread and places the pre-empted thread at the head of its priority level queue in the dispatcher 11 (step s4).

[0035] If, at step s1, a new thread is not ready or, at step s2, no thread is running, the cpu secondary scheduler 16 determines whether the current time slice has

expired (step s5). If the current time-slice has not expired then the process returns to step s1 else dispatches the thread (step s7) at the head of the dispatcher queue for the highest priority greater than 15 (step s6).

[0036] If, at step s6, no threads with priorities greater than 15 are pending, the cpu secondary scheduler 16 inspects the next element of the master signature (step s8). If the element is empty, the cpu secondary scheduler 16 dispatches the thread from the front of the highest priority queue that is not empty (step s9). If the element is not empty, the cpu secondary scheduler 16 looks for a thread belonging to the application, identified in the master signature element, in the highest priority populated queue in the dispatcher 11 (step s10). If one is found then it is dispatched (step s11) else the thread from the front of the queue is dispatched (step s12) and the cpu secondary scheduler 16 attempts to move the reservation (step s13). This dynamic slot shift can only be made to the extent of the next slot reserved by the same guaranteed service application 21. The reservation is returned to its correct position for the next cycle through the master signature and if the master signature is updated by the cpu reservation component 13.

[0037] It should be noted that whenever a guaranteed service thread cannot be serviced at steps s8 and s10, the best-effort service application 25 has the possibility of having its threads 26 dispatched.

[0038] When a component 22 of the guaranteed service application 21 is destroyed, its destructor 27 it may invoke a reservation cancel method of the primary scheduler 12, passing the reservation ID as a parameter. The primary scheduler 12 then invokes a cancellation method of the cpu reservation component 13 which removes the calling component's slot reservations from the copy 31 of the master signature 32 and then updates the master signature 32.

[0039] Considering now the case of disk access, an IRP (I/O Request Packet) is an encapsulated request, which defines the processing requirements, including the type of operation to be carried out (read or write), the number of bytes to manipulate, the synchrony of the request etc..

[0040] The constructor 24 of a component 22 includes a procedure for registering the guaranteed service application 21 for disk access. This process calls a disk access reservation method of the primary scheduler 12 with a demand level indicator, usually a percentile, as a parameter. The primary scheduler 12 then calls a disk access reservation method of the hard disk drive reservation component 14 with the demand level indicator as a parameter. The hard disk drive reservation component 14 then allocates a number of "lottery tickets" to the application 21 containing the component 22 in dependence on the demand level indicator. The component 22 may request an abstract reservation in terms of a class identifier which the primary scheduler 12 can use to reference a persistent lookup table mapping the appropriate demand level indicator.

[0041] When a thread 23, belonging to a guaranteed service application 21, requires disk access, it sends an IRP to an operating systems I/O manager 20, which is then forwarded to the hard disk secondary scheduler 17 via the operating system's file system component 30. The operating system's file system component 30 increases the granularity of the access request, e.g. from a request for one large block a data to many small blocks of data. When the IRP reaches the hard disk secondary scheduler 17, it is place in a wait array according to the guaranteed service application's process ID. The hard disk drive secondary scheduler 17 runs a lottery by randomly generating "ticket" numbers. If winning application 21 has an IRP in the wait array, the winning application 21 is granted disk access and the longest waiting IRP for the winning application is passed on to a physical disk device driver process 32. When there is no winning ticket holder or the winning ticket holder does not have an IRP ready for servicing, then an alternative IRP is scheduled via a simple cyclic selection scheme which includes IRPs from the best effort service application 25.

[0042] When a component 22 of the guaranteed service application 21 is destroyed, its destructor 27 determines whether it is the only instance of a component 22 of the application 21. If it is the last such instance, it invokes a cancel hard disk access method of the primary scheduler 12 which in turn invokes a cancellation method of the hard disk drive reservation component 14. The hard disk drive reservation component 14 then de-allocates the tickets allocated to the application 21.

[0043] Considering now the case of memory access, each application 21, 25 maintains its own unique virtual address space, which is a set of memory addresses available for its threads to use. This address space is much larger than the RAM 2 of the computer. When a component 22, 26 references its application's virtual memory, it does so with a pointer which is used by a virtual memory manager to determine the location in RAM 2 to which the virtual address maps. However, some portions of memory may actually reside on disk 10 or backing store. This means that data or code which is infrequently accessed is held on disk 10 thus retaining RAM 2 for better use. To facilitate this scheme, memory is managed in fixed size blocks known as "pages", which are usually 4Kb or 8Kb. If a component 22 references an address whose page is not in RAM 2, then a page fault occurs. This triggers a process known as "paging" which is the task of locating the faulted page within the page file and then loading the page into RAM 2. If there is insufficient space in RAM 2, then a page must be first swapped out and written to the page file. However, each process usually keeps a minimum number of pages locked into RAM 2 known as the "working set".

[0044] All modern operating systems generally protect memory in three forms:- physical hardware disallows any thread from accessing the virtual address space of other components, a distinction is made between the mode of operation, kernel-mode (ring 0)

which allows threads access to system code and data, and user-mode (ring 3) which does not, and a page-based protection mechanism wherein each virtual page maintains a set of flags that determine the type of access permitted. However, these mechanisms are aimed at protecting a component's memory resources from unwanted interference by other unauthorised processes in the system. They do not prevent a process from consuming more than its reasonable share of memory.

[0045] To aid understanding of the approach of the present invention to the implementation of working set size control, the working set management scheme in Windows NT will be briefly described. When created, each component is assigned two thresholds, a minimum working-set size and maximum working-set size. The minimum defines the smallest number of pages the virtual memory manager attempts to keep locked concurrently in physical memory, whilst the maximum defines an expansion range which can be used if the component is causing a considerable number of page faults. If the working set is too small, then the component incurs a large number of paging operations and thus a substantial overhead is incurred through continuously swapping pages to and from the backing store. Alternatively, if the working set is too large, few page faults occur but the physical memory may be holding code and/or data which is infrequently referenced and thus overall efficiency is reduced. In Windows NT, the Memory Manager (MM) adjusts the working sets once every second, in response to page-in operations or when free memory drops to below a given threshold. If free memory is plentiful, the MM removes infrequently referenced pages only from working sets of components whose current size is above a given minimum, this is known as aggressive trimming. However, if free memory is scarce, the MM can force trimming of pages from any component until it creates an adequate number of pages, even beyond the minimum threshold. Of course, components which have extended working sets are trimmed in preference to those which have the minimum working set size.

[0046] In the present embodiment, the constructor 24 of a component 22 includes a procedure for reserving a working set. This procedure calls a memory reservation method of the primary scheduler 12 which, in turn calls a method of the memory reservation component 15. The memory reservation component 15 translates this request into the appropriate expansion of a default minimum working set size. The minimum working set size regulates the number of pages an application 21 can concurrently lock (or pin) into RAM 2. Thus, a component 22 that wishes to reserve a portion of memory, makes the reservation request via the primary scheduler 12 to the memory reservation component 15 which then increases the working set thresholds accordingly. This adjustment is facilitated through operating system calls. It is then the responsibility of the component to pin the pages into RAM 2 as required. The pinning process is arbitrated on a first-come first-serve basis. If insufficient physical pages are available, the operating system will reject the pin request, therefore the component 22 is expected to pin its reserved pages soon after allocation.

[0047] If there are insufficient free physical memory resources, the memory reservation component 15 then attempts to force the processes of best-effort service applications 25 to write pages back to file store until sufficient free physical pages are available. If insufficient space can be made, then the reservation is rejected. As with the other resource modules, a portion of the RAM 2 is reserved for a minimal best-effort service. This is facilitated by maintaining a count of the total number of physical pages being used under the guaranteed service and checking this against a portion of the total physical page capacity. It should also be noted that the 'best-effort' service is strictly a minimal guaranteed service (defined by the system assigned minimum working set size) in addition to the best-effort service through potential expansion of the working set. It is assumed that processes cannot bypass the primary scheduler 12 and directly alter their own working set size. Finally, special consideration is made for pages which are being used for shared memory purposes. If any client of a shared page has undergone reservation, then the page is classified as reserved even though other clients may not have made a reservation.

[0048] A second embodiment is the same as the first embodiment described above except for the manner of operation of the cpu secondary scheduler 16. The operation of the cpu scheduler of the second embodiment will now be described.

[0049] Referring to Figures 2 and 4, the cpu secondary scheduler 16 repeatedly performs the following process. The cpu secondary scheduler 16 determines whether a new thread is pending in the dispatcher 11 (step s101). If a new thread is pending, the cpu secondary scheduler 16 determines whether a thread is currently running (step s102). If a thread is currently running, it determines whether the new pending thread has a higher priority than the running thread (in this context a guaranteed service thread has "priority" over a best-effort service thread with the same priority) (step s103). If the new thread's priority is higher, the cpu secondary scheduler 16 pre-empts the running thread, dispatches the new thread and places the pre-empted thread at the head of its priority level queue in the dispatcher 11 (step s104).

[0050] If, at step s101, a new thread is not ready or, at step s102, no thread is running, the cpu secondary scheduler 16 determines whether the current time slice has expired (step s105). If the current time-slice has not expired then the process returns to step s101 else dispatches the thread (step s107) at the head of the dispatcher queue for the highest priority greater than 15 (step s106).

[0051] If, at step s106, no threads with priorities greater than 15 are pending, the cpu secondary scheduler 16 inspects the next element of the master signature (step

s108). If the element is empty, the cpu secondary scheduler 16 dispatches the thread from the front of the highest priority queue that is not empty (step s109). If the element is not empty, the cpu secondary scheduler 16 looks for a thread belonging to the application, identified in the master signature element, in a queue in the dispatcher 11 (step s110), searching from the priority 15 queue towards the lowest priority queue and from the head of each queue to the tail thereof. If one is found then it is dispatched (step s111) else the thread from the front of the highest priority populated queue is dispatched (step s112) and the cpu secondary scheduler 16 attempts to move the reservation (step s113). This dynamic slot shift can only be made to the extent of the next slot reserved by the same guaranteed service application 21. The reservation is returned to its correct position for the next cycle through the master signature and if the master signature is updated by the cpu reservation component 13.

[0052] It should be noted that whenever a guaranteed service thread cannot be serviced at steps s108 and s110, the best-effort service application 25 has the possibility of having its threads 26 dispatched.

[0053] The present invention has been described with reference to applications comprising components which in turn comprise one or more threads. It will be appreciated that the present invention is not limited to this situation and may be implemented in a system in which application programs are not built using object-oriented languages.

[0054] Whilst the present invention has been described with reference to cpu, hard disk and memory access, it will be appreciated that it may be applied to the scheduling of access to other resources.

[0055] It will be appreciated that the term "lottery ticket" is used figuratively.

Claims

1. A method of administering resource utilisation in a computer, the method comprising running a first process to make a reservation for access to a resource in dependence on a resource requirement communication from an application process and running a second process to grant requests for access to said resource from said application process in dependence on said reservation.

2. A method according to claim 1, comprising:

   instantiating a scheduling object having a method or methods for processing reservation requests for a plurality of resources and initiating resource specific reservation processing;
   instantiating a reservation object having a method or methods for making reservations for access to a resource;
   invoking a method of the scheduling object from an application process, said method taking a first resource access requirement definition as a parameter;
   invoking, from said method of the scheduling object with a second resource access requirement definition as a parameter, a reservation method of the reservation object to make a reservation for the application process;
   running a resource specific scheduling process to grant access to a resource in dependence on the reservation made by the reservation object; and
   utilising said resource for the purposes of said application process.

3. A method according to claim 2, wherein said method of the scheduling object translates the first resource requirement definition into the second resource requirement definition.

4. A method according to claim 1, 2 or 3, wherein said resource is a cpu.

5. A method according to claim 1, 2 or 3, wherein said resource is a mass storage device.

6. A method of scheduling access to a cpu, the method comprising the steps of:

   generating a one-dimensional reservation request pattern;
   merging the reservation request pattern with a one-dimensional cpu access control pattern, representing empty cpu access time slots and reserved cpu access time slots, without substantially disturbing either the reservation request pattern or the reserved cpu access time slots in the reservation request pattern.

7. A method according to claim 6, wherein said merging step comprises relocating a non-empty time slot element of the reservation request pattern or the cpu access control pattern such that the patterns can be merged without any reserved cpu access time slot elements being deleted or overwritten.

8. A method according to claim 7, wherein the relocated non-empty time slot element is relocated by an amount defined in said time slot element.

9. A method according to claim 1, wherein said first process is a method according to claim 6, 7 or 8.

10. A method of granting access to a cpu comprising:

    generating a one-dimensional cpu access control pattern, each element of which relates to a

quantum of cpu access time; and
at the end of a quantum of cpu access time:

granting access to any pending processes having a priority greater than a predetermined level; and then
if the next pattern element is empty then granting access to a pending process meeting a predetermined prioritisation criterion else granting access to a process identified in the pattern element.

11. A method according to claim 10, wherein pending processes populate queues having different priorities and access is granted to the process identified in the pattern element when there is not a populated process queue having a higher priority than the queue in which said process is present.

12. A method according to claim 10 or 11, wherein the a one-dimensional cpu access control pattern is generated by a method according to claim 6, 7 or 8.

13. A method according to claim 2, wherein the second process performs a method according to claim 10 or 11.

14. A method according to claim 1, wherein said first process comprises allocating one or more lottery ticket numbers to said application process in dependence on the second resource access requirement definition and the second process performs a method comprising the steps of:

storing requests for access to a mass storage device from application processes;
generating substantially randomly a lottery ticket number; and
if no application process has been allocated said lottery ticket number then passing on to a mass storage device driver process the stored request for access from an application process selected on the basis of a predetermined prioritisation criterion else passing on to a mass storage device driver process a stored request for access from an application process to which said lottery ticket number was allocated.

15. A computer configured to operate in accordance with a method according to any preceding claim.

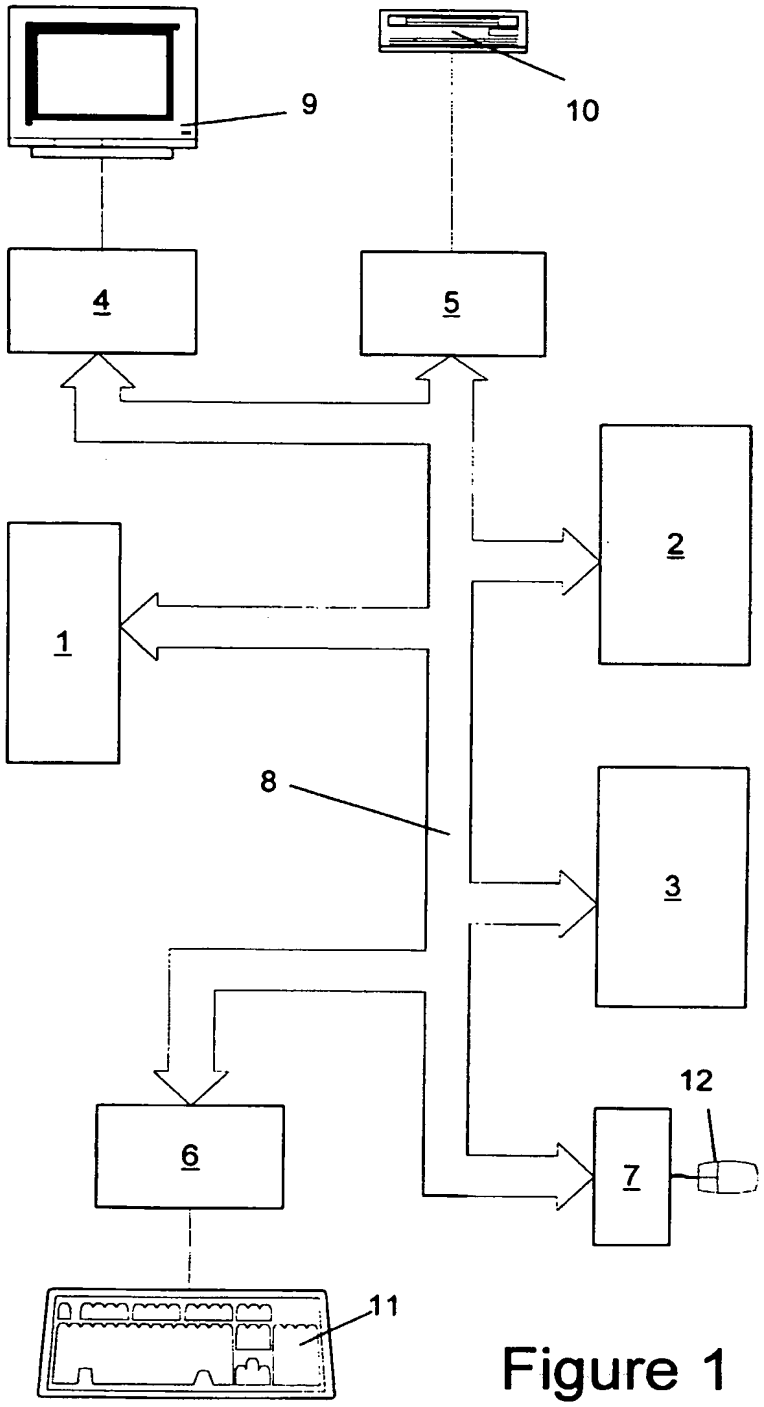

Figure 1

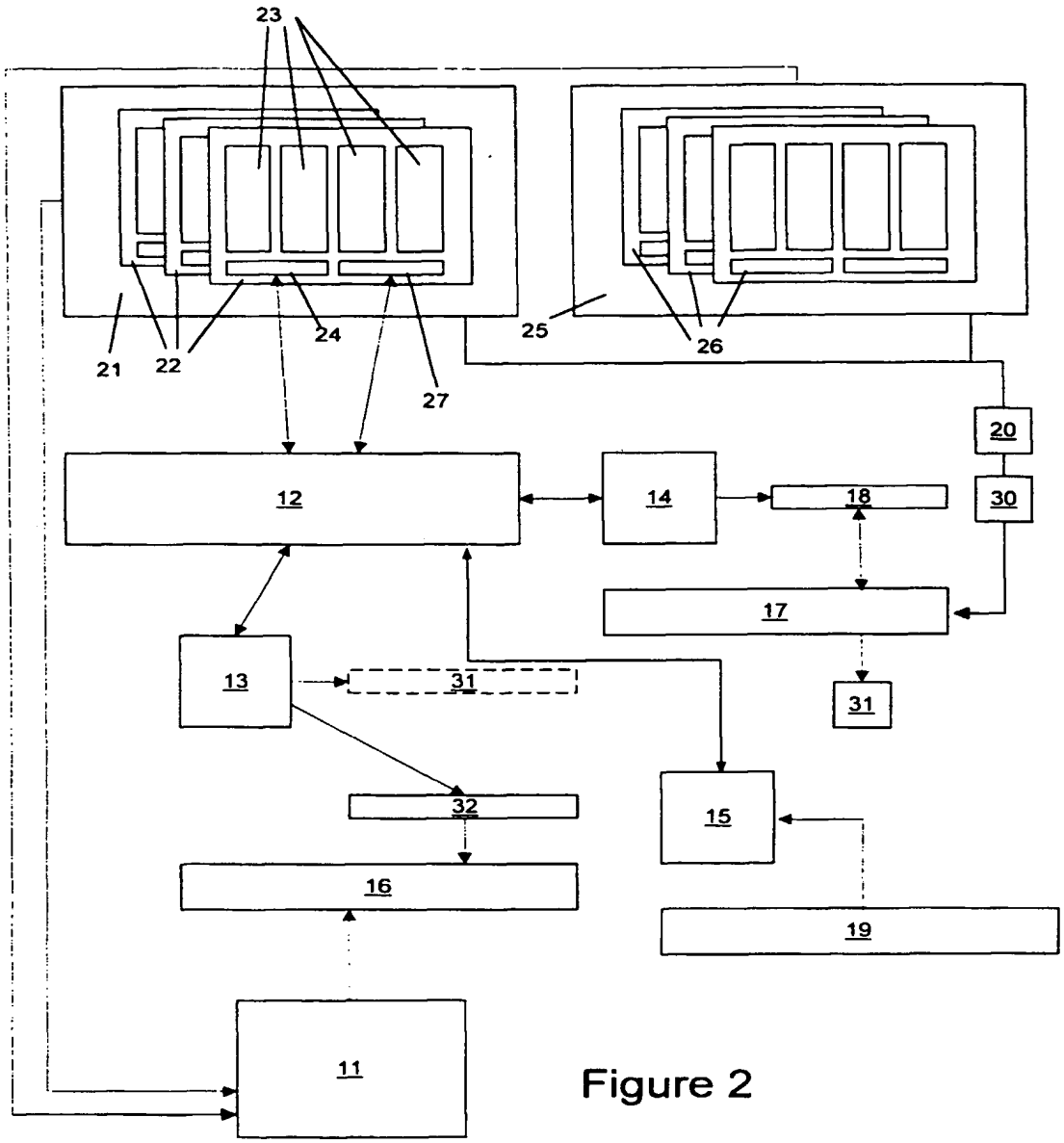

Figure 2

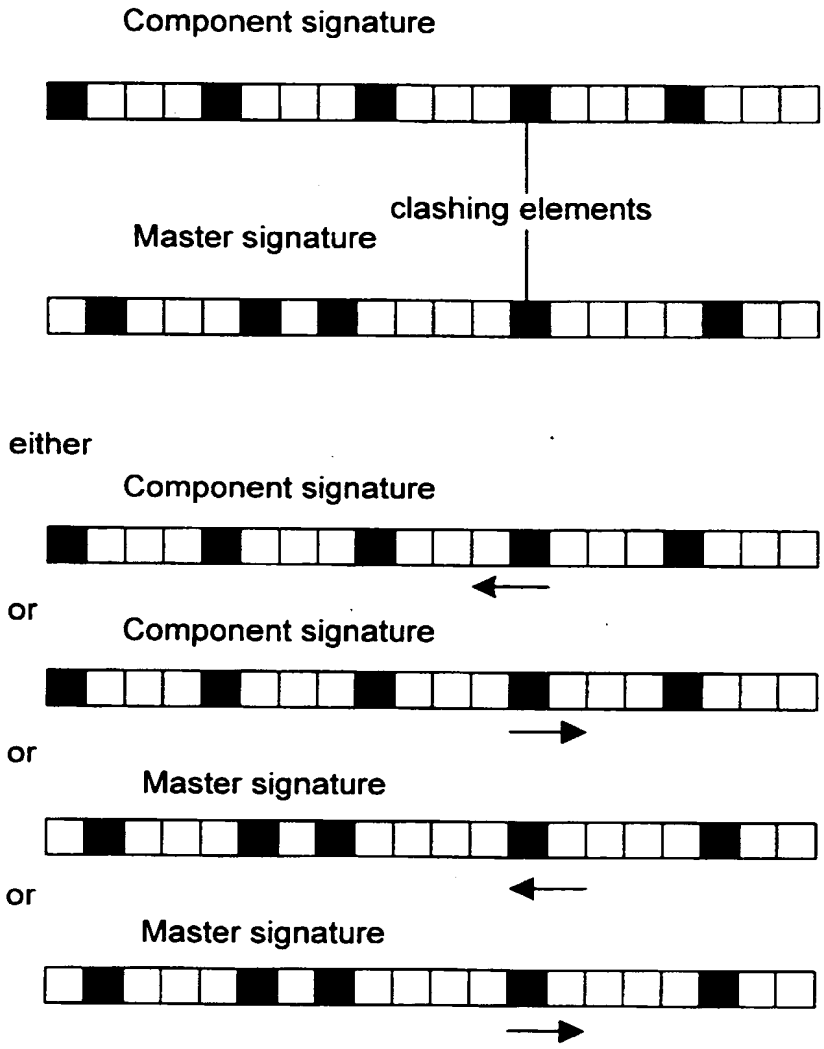
Component signature
clashing elements
Master signature
either
Component signature
or
Component signature
or
Master signature
or
Master signature

Figure 3

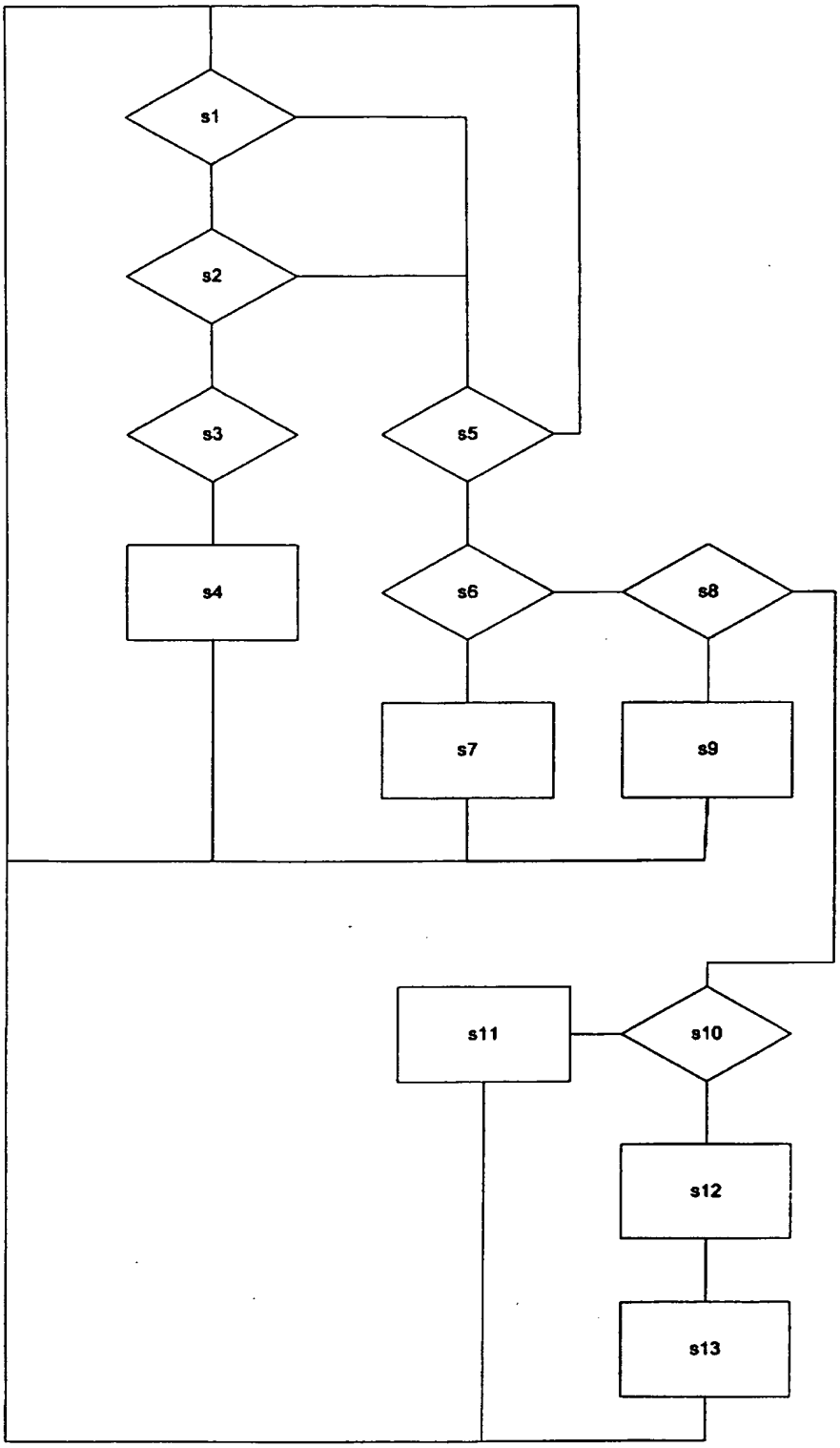

Figure 4

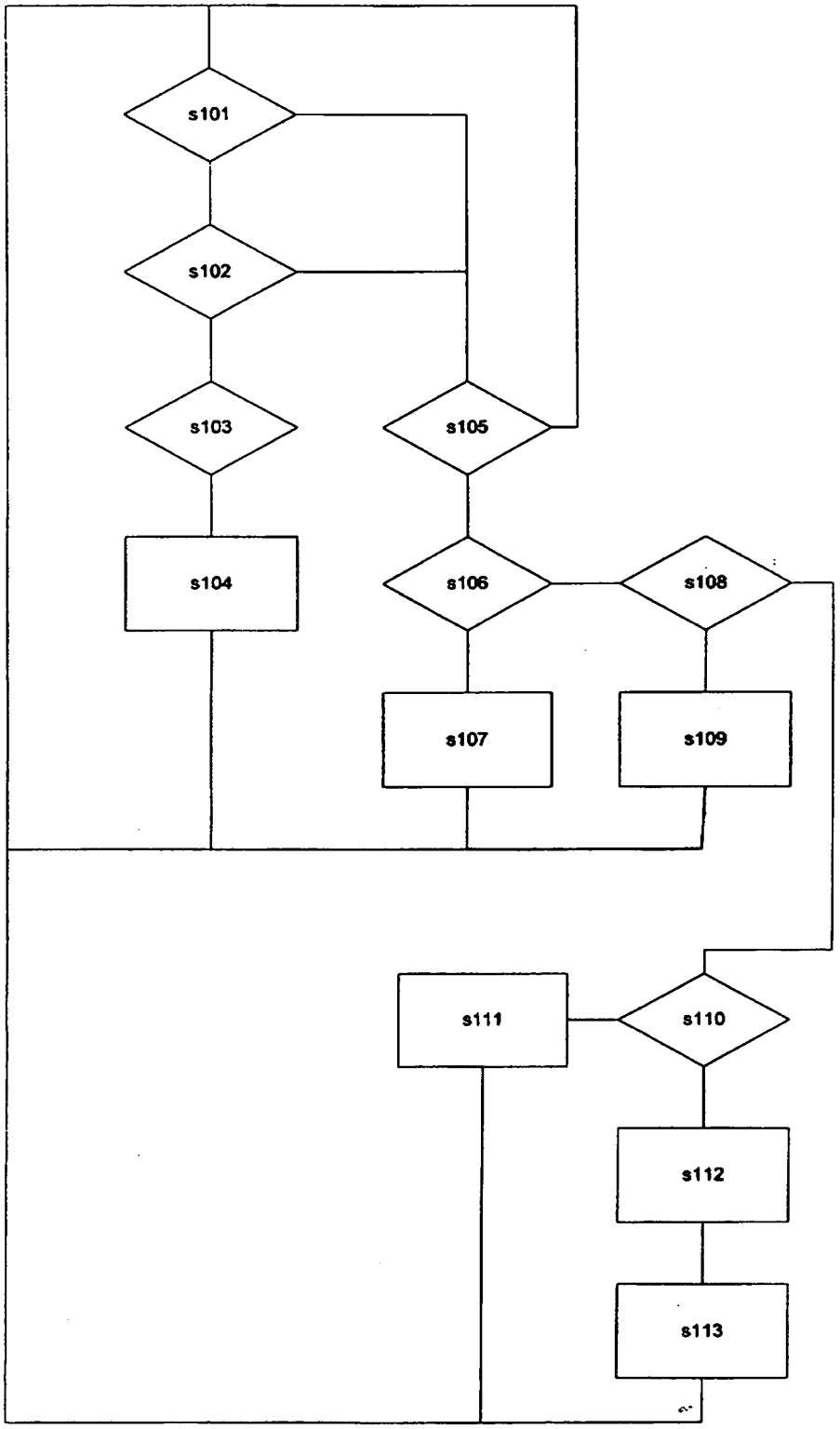
s101
s102
s103
s104
s105
s106
s107
s108
s109
s110
s111
s112
s113

Figure 5

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 30 3936

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
X A	EP 0 798 638 A (HITACHI LTD) 1 October 1997 (1997-10-01) * column 8, line 22 - column 14, line 20 * * column 32, line 56 - column 33, line 24 * ---	1,4,6,7, 9,15 2,3,5, 10,12,13	G06F9/46
X A	US 5 812 844 A (DRAVES JR RICHARD P ET AL) 22 September 1998 (1998-09-22) * column 6, line 56 - column 10, line 23 * * claims 1-3 * ---	1,3,4,15 2,5-10, 12,13	
X A	EP 0 817 041 A (SUN MICROSYSTEMS INC) 7 January 1998 (1998-01-07) * column 2, line 54 - column 5, line 31 * * claims 1-7 * ---	1,3 2,4,5	
A	US 5 247 677 A (WELLAND ROBERT V ET AL) 21 September 1993 (1993-09-21) * column 5, line 24 - column 6, line 53 * ---	1,14	TECHNICAL FIELDS SEARCHED (Int.Cl.7)
A	EP 0 790 557 A (MATSUSHITA ELECTRIC IND CO LTD) 20 August 1997 (1997-08-20) * page 8, line 22 - page 13, line 38 * ---	1,14	G06F
A	EP 0 658 841 A (IBM) 21 June 1995 (1995-06-21) * column 5, line 34 - column 7, line 8 * -----	1,10	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	29 September 1999	Bijn, K

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 99 30 3936

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

29-09-1999

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0798638 A	01-10-1997	CA 2200929 A	28-09-1997
		JP 2904483 B	14-06-1999
		JP 9319597 A	12-12-1997
US 5812844 A	22-09-1998	NONE	
EP 0817041 A	07-01-1998	US 5826082 A	20-10-1998
		JP 10063519 A	06-03-1998
US 5247677 A	21-09-1993	FR 2691557 A	26-11-1993
		JP 6035726 A	10-02-1994
EP 0790557 A	20-08-1997	JP 9282184 A	31-10-1997
EP 0658841 A	21-06-1995	US 5487170 A	23-01-1996
		JP 7200318 A	04-08-1995

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82